Doom Reinforcement Learning - Raport

Adam Kaniasty, Hubert Kowalski, Norbert Frydrysiak, Igor Kołodziej, Krzysztof Sawicki

11 lipca 2024

Streszczenie

Przedstawiony raport opisuje implementację i badania wydajności modelu mającego na celu przejście scenariuszy **ViZDoom**[2]. Proces trenowania agenta odbywał się przy użyciu dwóch metod uczenia maszynowego: Proximal Policy Optimization (PPO) oraz Advantage Actor-Critic (A2C). Obie te metody opierają się na uczeniu przez wzmocnienie, gdzie agent uczy się poprzez interakcję ze środowiskiem oraz otrzymywanie nagród za wykonywane działania. Raport obejmuje opis implementacji samego modelu oraz wyniki przeprowadzonych badań, które obejmowały dobranie odpowiednich parametrów funkcji nagród oraz sposobu i długości uczenia, aby agent pokonał poziom w najefektywniejszy sposób. W ramach badań przeprowadzono bezpośrednie porównanie wydajności modeli A2C i PPO, z którego wynika, że metoda PPO osiągnęła lepsze wyniki.

Spis treści

1 Wstęp

W niniejszym raporcie przedstawiamy wyniki projektu z przedmiotu "Warsztaty Badawcze". Naszym celem było rozwiązanie gry Doom przy użyciu metod z dziedziny Reinforcement Learning. W tym celu stworzyliśmy dwa modele: (A2C oraz modyfikowany PPO) oraz stworzyliśmy dwie funkcje nagród. W trakcie badań przeprowadziliśmy testy wydajności aby wyłonić najlepszy zestaw.

1.1 Opis koncepcji algorytmu

Reinforcement learning (RL) [5] to jedna z metod uczenia maszynowego, która polega na tym, że agent uczy się poprzez interakcję ze środowiskiem i otrzymywanie nagród za wykonywane działania. Główne elementy RL to:

- Agent: Program lub algorytm, który podejmuje działania.
- Środowisko: Wszystko, z czym agent wchodzi w interakcję.
- Działania (actions): Możliwe decyzje lub ruchy, które agent może wykonać.
- Stan (state): Bieżąca sytuacja lub konfiguracja środowiska.
- Nagrody [1] (rewards): Informacja zwrotna ze środowiska, która mówi agentowi, jak dobre było jego działanie.
- Polityka (policy): Strategia, która określa, jakie działanie agent powinien podjąć w danym stanie.

Agent RL skupia się na maksymalizacji sumy z funkcji nagród, które otrzymuje przez cały czas trwania gry. Proces uczenia obejmuje modyfikowanie polityki w oparciu o zaobserwowane postępy.

1.2 ViZDoom

ViZDoom to narzędzie i platforma badawcza wykorzystywana do trenowania i testowania algorytmów sztucznej inteligencji, w szczególności w kontekście Reinforcement Learning. ViZDoom umożliwia agentom AI granie w grę Doom dostarczając przy tym środowisko 3D odzwierciedlające stan gry. Rozgrywka opiera się na dostarczonym silniku gry, który umożliwia między innymi poruszanie się w trójwymiarze. ViZDoom posiada interfejs API w języku Python, co ułatwia integrację z narzędziami i bibliotekami uczenia maszynowego, takimi jak TensorFlow oraz PyTorch.

[1] Nagrody mogą być pozytywne (za dobre decyzje) lub negatywne (za złe decyzje)

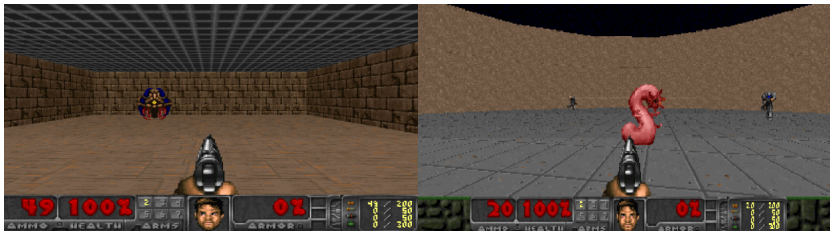
1.3 Opis gry

"Doom" to klasyczna gra komputerowa typu first-person shooter (FPS), która została stworzona przez id Software i po raz pierwszy wydana 10 grudnia 1993 roku. Jest to jedna z najbardziej wpływowych gier w historii branży gier komputerowych, która miała ogromny wpływ na rozwój gatunku FPS. W naszym projekcie naszym głównym scenariuszem był poziom "Death Corridor"[2], który jest często wykorzystywany jako benchmark w badaniach nad uczeniem ze wzmocnieniem. Charakteryzuje się on:

- Wąskim korytarzem - Gracz porusza się w korytarzu co uniemożliwia omijanie wrogów
- Duża liczba wrogów - ich liczba może być zmieniana co wpływa na stopień trudności
- Ograniczona ilość zasobów - gracz dysponują skończoną liczbą amunicji oraz życia

Celem poziomu jest dotarcie na koniec korytarza. Poziom trudności może zostać zmienany przy pomocy parametru. Jednocześnie przetestowaliśmy nasz model na scenariuszu "Basic" oraz "Defend Center". Pierwszy z nich polegał na trafieniu wroga znajdującego się naprzeciwko agenta. W drugim scenariuszu przeciwnicy zbliżali się do agenta, a on obracając się musiał ich zabić nim do niego dojdą.

Rysunek 1: Zrzut ekranu z rozgrywki Death Corridor.



Rysunek 2: Zrzut ekranu z rozgrywki Basic oraz Defend Center.

2 Algorytmy

Algorymy PPO (Proximal Policy Optimization) [1] oraz A2C (Advantage Actor-Critic) [4] skupiają się głównie na się architekturze sieci neuronowej używanej do reprezentowania sieci polityki i funkcji wartości [3].

1. **Polityka (Actor):** Sieć neuronowa, która przyjmuje stan środowiska i wyprowadza rozkład prawdopodobieństwa akcji. Ten rozkład reprezentuje politykę agenta, która określa prawdopodobieństwa podjęcia każdej akcji w danym stanie.
2. **Funkcja wartości (Critic):** Sieć neuronowa, która przyjmuje stan i przewiduje wartość tego stanu, czyli oczekiwaną skumulowaną nagrodę.

[2]Poziom ten został stworzony przez społeczność graczy, a nie przez główne wydawnictwo

Owe algorytmy działają w podobny sposób, jednakże PPO rozwija koncepcje aktualizacji polityki A2C i stosuje klipowanie [6], aby zapewnić stabilność treningu.

2.1 Działanie algorytmu

1. **Inicjalizacja:** Inicjalizujemy dwie sieci neuronowe: jedną dla polityki (actor) i drugą dla funkcji wartości (critic).

2. **Interakcja ze środowiskiem:** Agent obserwuje bieżący stan s_t. Polityka (actor) wybiera akcję a_t na podstawie rozkładu prawdopodobieństwa wyprowadzonego z bieżącego stanu s_t.

3. **Wykonanie akcji:** Akcja a_t jest wykonywana w środowisku, co prowadzi do nowego stanu s_{t+1} i nagrody r_{t+1}.

4. **Obliczenie wartości:** Funkcja wartości (critic) ocenia stan s_t i przewiduje wartość $V(s_t)$. Funkcja wartości ocenia również nowy stan s_{t+1} i przewiduje wartość $V(s_{t+1})$.

5. **Obliczenie Advantage:** Obliczamy advantage $A(s_t, a_t)$, który jest różnicą między otrzymaną nagrodą plus wartością przyszłego stanu a wartością bieżącego stanu:

$$A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

   gdzie γ jest współczynnikiem dyskontowania.

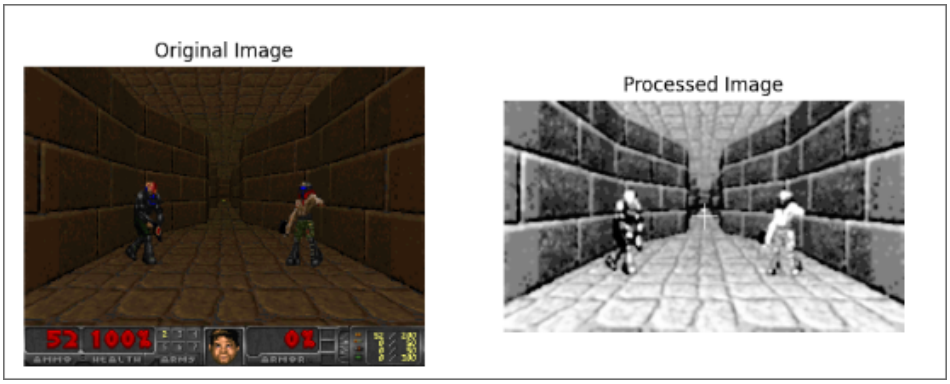
6. **Aktualizacja sieci:**

   - **Aktualizacja polityki (A2C):** Gradient polityki jest aktualizowany na podstawie advantage:
   $$\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$
   gdzie π_θ jest polityką parametrów θ.
   - **Aktualizacja polityki (PPO):** Aktualizujemy politykę z ograniczeniem za pomocą klipowania [1]:
   $$L^{CLIP}(\theta) = E_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$
   gdzie:
     - $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ - stosunek nowych i starych prawdopodobieństw,
     - ϵ - parametr decydujący o stopniu clipowania.
   - **Aktualizacja funkcji wartości:** Aktualizujemy funkcję wartości na podstawie błędu średniokwadratowego (MSE):
   $$\text{MSE} = (r_t + \gamma V(s_{t+1}) - V(s_t))^2$$

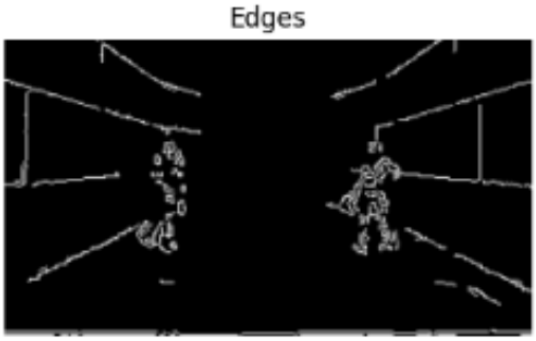
7. **Powtarzanie kroków:** Kroki 2-6 są powtarzane przez określoną liczbę epok lub kroków treningowych.

2.2 Parametry sieci neuronowej

Podczas naszych badań przestrzeń stanów składała się z danych sekwencyjnych (dane z gry) oraz strukturalnych (obraz z gry) [4]. Z tego powodu użyliśmy sieci sieci typu CNN (Convolutional Neural Network) zawierającej przystosowane do tego warstwy konwolucyjne. Za funkcję aktywacji została przyjęta funkcja ReLU. Na wyjściu znajduję się warstwa liniowa. W celu poprawy efektywności uczenia agenta, wprowadziliśmy wstępne przetwarzanie obrazu polegający na usunięciu zbędnych informacji o stanie gry. Obraz został przekształcony na skale szarości oraz przeskalowany. Pasek z informacjami o stanie gry został przycięty, aby wyeliminować nieistotne dane. Dodatkowo wygenerowaliśmy wersję obrazu zawierającą wyłącznie krawędzie, co miało na celu wyeksponowanie istotnych elementów sceny.



Rysunek 3: Wstępne przetwarzanie obrazu z gry.



Rysunek 4: Obraz z gry z wyróżnionymi krawędziami.

2.3 Hiperparametry

Podczas procesu uczenie wyróżniamy cztery hiperparametry, które istotnie wpływały na przebieg oraz tempo trenowania. Należą do nich (obrane wartośći w nawiasach:

- N Steps (4096 lub 8192) - odstęp czasowy regulujący aktualizację sieci.
- Gamma (0.95) - współczynnik dyskonta dla nagrody, jego podwyższenie wymusza na agencie podejmowanie czynności dających długofalowe korzyści
- Clip range (0.01) - zapobiega bardzo dużym zmianom w polityce
- Entropy Regularization (0.05) - wprowadza penalizacje za akcje których jest bardzo pewny. Dzięki temu zapobiegamy przedwczesnym zachowaniom deterministycznym.

3 Funkcja nagrody

Zaimplementowana funkcja nagrody zwracała uwagę na kilka czynników. Docelowo model miał za zadanie zebrać zbroję znajdującą się na końcu korytarza. Przed tym powinien on zabić przeciwników w korytarzu, aby nie stracić za dużo życia. Adresowane zmiany stanu gry:

- Zdrowie - agent był karany za stratę zdrowia
- Amunicja - agent był karany za wykorzystanie zbyt dużej ilości amunicji, miało to go zniechęcić do ciągłego strzelania
- Kills - agent był nagradzany za każde zabójstwo, aby zachęcić go do walki z przeciwnikami
- X change - agent otrzymywał nagrody za poruszanie się do przodu. My jednak, w ramach naszej nagrody, wprowadziliśmy karę za tę czynność, aby zrównoważyć wbudowany system nagradzania i umożliwić skalowanie wszystkich nagród. Naszą nagrodę dodawaliśmy do nagrody domyślnej. Zazwyczaj agent otrzymywał sumaryczną dodatnią nagrodę za chodzenie do przodu, natomiast za poruszanie się do tyłu otrzymywał nagrodę ujemną, której wartość bezwzględna była równa nagrodzie za chodzenie do przodu. Nagroda była wyliczana na podstawie zmiany współrzędnych na osi OX. W dalszych rozdziałach pracy będziemy podawać sumaryczną nagrodę za tę czynność. Jeśli będzie dodatnia, to znaczy, że nagradzamy za chodzenie do przodu i karamy za chodzenie do tyłu.

Dodatkowo agent otrzymywał nagrodę za zakończenie poziomu poprzez zebranie zbroi. Konkretne wartości nagród była dobierana eksperymentalnie na podstawie obserwacji procesu uczenia.

4 Metryki

Aby zweryfikować poprawność procesu uczenia zostały wprowadzone metryki mające na celu monitorować postępy. Wraz z obserwacją rozgrywki dawały one pełny wgląd na umiejętności modelu. Wprowadzone metryki są następujące:

- Episode Ammo - mierzy ilość amunicji użytej w każdym epizodzie.
- Episode Distance - mierzy przebyty dystans od punktu (0,0,0) w przestrzeni trójwymiarowej w każdym epizodzie.
- Episode Health - mierzy ilość zdrowia na końcu epizodu. Metryka zwraca wartość różną od zera w dwóch przypadkach:
  - agent dotarł do końca korytarza i zebrał zbroje
  - upłynął czas przeznaczony na epizod
  W przeciwnym wypadku agent umierał, a jego zdrowie wynosiło 0.
- Episode Killcount - zlicza ilość przeciwników zabitych podczas jednego epizodu. Docelowo chcemy osiągnąć wartość 6, gdyż tyle wrogów znajduję się w całym korytarzu.
- Episode Return - mierzy zwrot otrzymywany przez agenta w każdym epizodzie. Jego wartość jest obniżana przez parametr gamma.
- Episode Steps - mierzy ilość kroków wykonaną w każdym epizodzie
- Timestep Reward - mierzy nagrody w każdym time stepie. Nagrody są dodawane do całkowitego zwrotu i skalowane przez czynnik gamma.

5 Proces trenowania

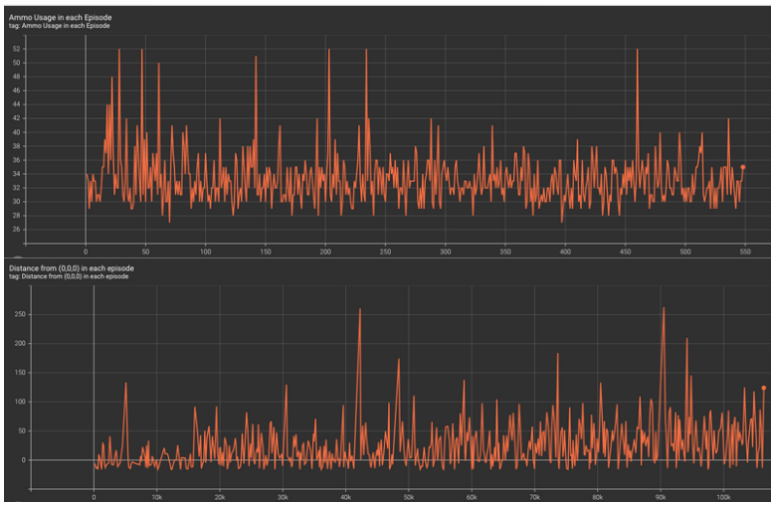
5.1 Wstępne uczenie

W początkowych fazach trenowania został użyty model PPO, uczony na 100k time stepach na poziomie trudności 2. Pod uwagę była brana jedynie nagroda customowa tzn. nagroda wbudowana w środowisku nie była dodawana do sumy. Parametry nagrody prezentowały się następująco:

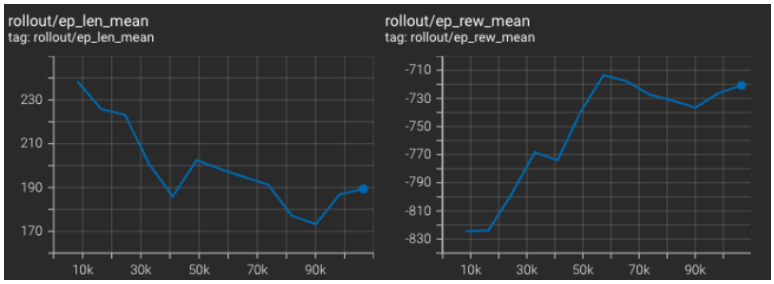
Parametr	Wartość
Stracony punkt życia	-11
Zabicie przeciwnika	210
Użyty nabój	-2
X change	+1
Śmierć	-100

Tabela 1: Tabela przedstawiająca parametry nagrody wykorzystanej w początkowych fazach trenowania modelu PPO w scenariuszu Death Corridor.

Podczas uczenia polityka była niedeterministyczna, przez co agent czasem decydował się na losowy ruch, natomiast w trakcie testowania patrzyliśmy głównie na politykę deterministyczną, czyli agent zawsze decydował się ruchy, których nauczył się, że są najlepsze. W rezultacie podczas procesu uczenia liczba epizodów rozegranych wyniosła 549, spośród których agent zginął 545 razy, zaś pozostałe 4 epizody zakończyły się upływem czasu przewidzianego na ukończenie poziomu. Zabijał on średnio 0.53 wroga na epizod. Natomiast podczas testowania, wysoka nagroda za chodzenie do przodu spowodowała tendencje do poruszania się do przodu nie zwracając uwagi na przeciwników. Zadawanie małych obrażeń przez przeciwników sprawiało, że agent docierał do końca korytarza co utwierdzało go w poprawności swojej strategii. Metryki również nie wskazały na prawidłowy proces uczenia.



Rysunek 5: Wykres wartości metryki *ammo usage* w zależności od numeru epizodu oraz metryki *distance from (0,0,0)* w zależności od liczby time stepów.



Rysunek 6: Wykres wartości metryk episode length oraz episode mean reward w zależności od iteracji uczenia.

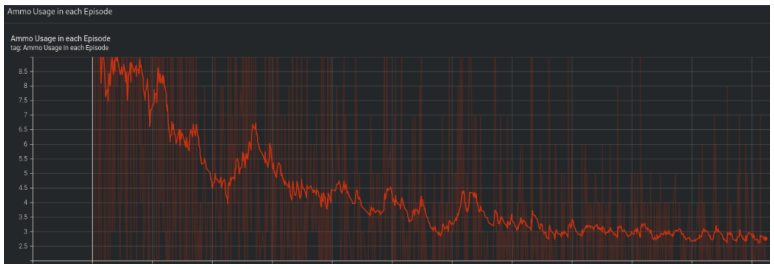
Kolejne fazy uczenia polegały na zmienianiu poszczególnych wartości w funkcji nagrody oraz dłuższym trenowaniu agenta, aby zachęcić go walki z przeciwnikami. W wyniku tej korekty agent zaczął nadużywać akcji strzelania powiązanej z zabiciem przeciwnika. Równocześnie przestał się poruszać do przodu i zaczął się skupiać tylko na dwóch pierwszych przeciwnikach.

5.2 Finalne rezultaty

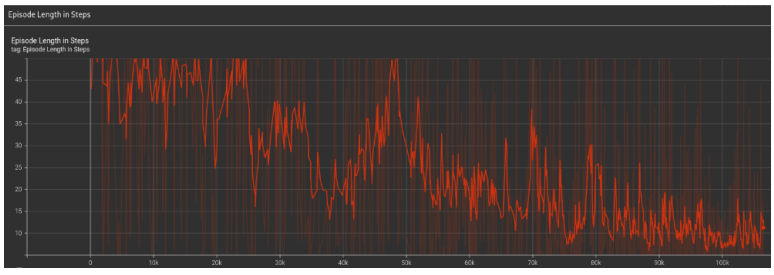
Rozwiązaniem, które przyniosło oczekiwany efekt była zmiana inputu dla agenta podczas trenowania. Zamiast informacji o stanie gry w postaci danych skalarnych oraz obrazów otrzymywał on same obrazy. Zmniejszyło to złożoność uczenia oraz ułatwiło agentowi rozpoznawanie wrogów. To podejście zostało przetestowane na wszystkich trzech scenariuszach.

5.2.1 Basic

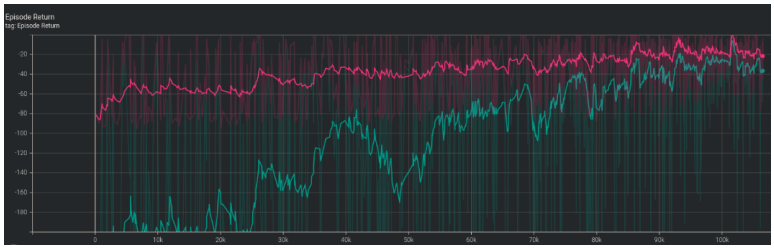
Obrany model PPO trenowany na 100k time stepach.



Rysunek 7: Wykres wartości metryki ammo usage w każdym epizodzie. Tendencja spadkowa sugeruje, że agent potrzebował coraz mniej kul do zabicia przeciwnika.



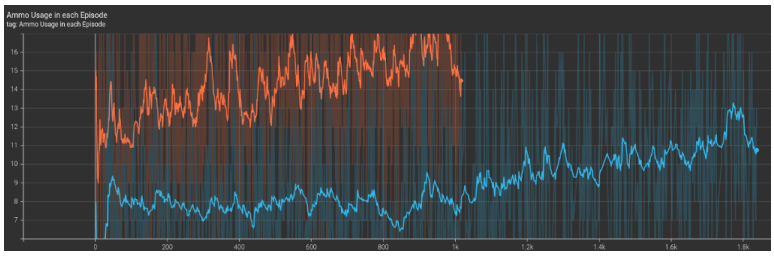
Rysunek 8: Wykres wartości metryk episode lengths w zależności od liczby time stepów. Spadek czasu episodu sugeruje, że agent szybciej zabija przeciwnika.



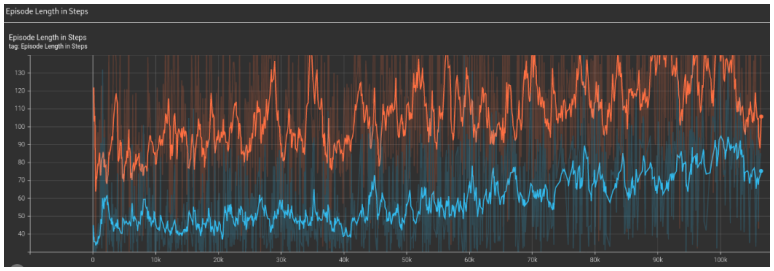
Rysunek 9: Wykres wartości metryki *episode return*. Różowa linia przedstawia wartość zdyskontowaną.

5.2.2 Defend center

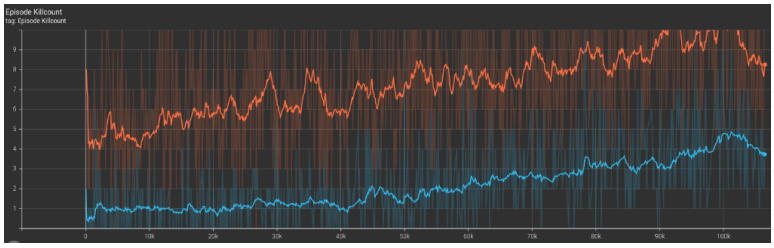
Użyliśmy tego samego modelu, wytrenowanego już na scenariuszu Basic. Do scenariusza defend the center został on dotrenowany o 200k time stepów. Na prezentowanych wykresach linia niebieska prezentuje trening podczas pierwszych 100k time stepów, a pomarańczowa następne 100k time stepów. Wykres ammo usage przedstawia ilość zużytej amunicji w danym epizodzie. Linia pomarańczowe przedstawia mniej pomiarów w tej metryce, co oznacza, że agent rozegrał w drugiej połowie treningu mniej epizodów, dłużej przeżywając.



Rysunek 10: Wykres wartości metryki ammo usage.



Rysunek 11: Wykres wartości metryki episode length.



Rysunek 12: Wykres wartości metryki killcount.

Jak możemy zauważyć na powyższych wykresach wszystkie metryki rosły z czasem. Wzrost ammo usage sugerował, że agent strzelał do większej liczby przeciwników co potwierdza metryka kill count.

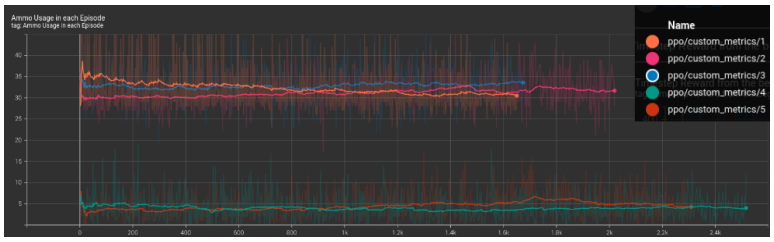
5.2.3 Death corridor

Oprócz samego procesu trenowania zmieniliśmy parametry funkcji nagrody w stosunku do wstępnego uczenia.

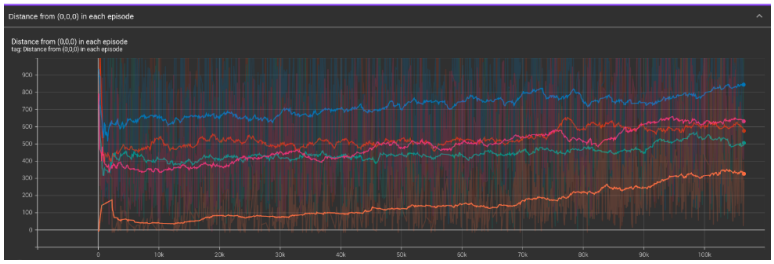
Parametr	Wartość
Stracony punkt życia	-1
Zabicie przeciwnika	200
Użyty nabój	-5
X change	+1
Śmierć	-200

Tabela 2: Tabela przedstawiająca parametry nagrody wykorzystanej podczas uczenia ostatecznych modeli.

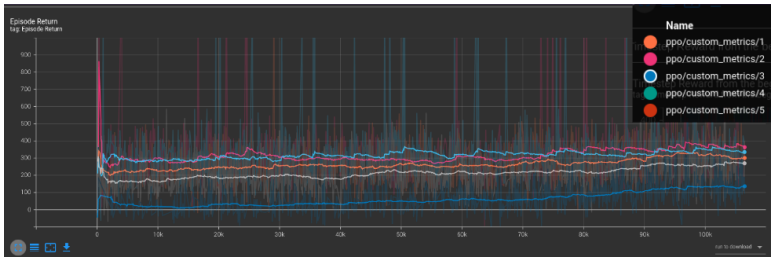
Model PPO trenowany był w 5 etapach: 100k, 200k, 300k, 400k, 500k. Oprócz PPO trenowany był również A2C, który poradził sobie znaczenie gorzej niż PPO. Z tego powodu jego uczenie zostało porzucone. Metryki prezentują się następująco (legendy we wszystkich wykresach oznaczają to samo, gdzie 1 oznacza 100k, 2 oznacza 200k itd):



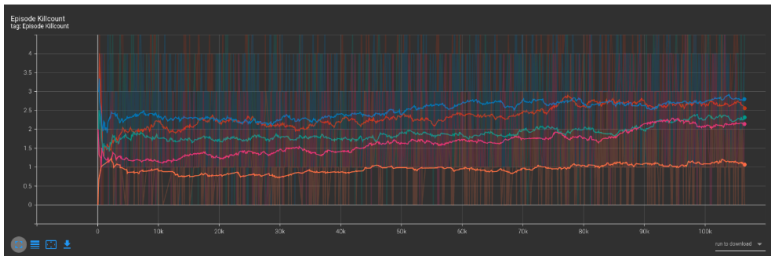
Rysunek 13: Wykres wartości metryki ammo usage. Długość wykresów różni się w zależności od etapu, ponieważ każdy etap trwa 100K kroków, a w jego trakcie może odbyć się różna ilosć epizodów. Na prostszych poziomach trudności epizody trwały średnio dłużej



Rysunek 14: Wykres wartości metryki episode distance.



Rysunek 15: Wykres wartości metryki episode return.



Rysunek 16: Wykres wartości metryki killcount.

6 Wnioski oraz podsumowanie

Wydłużenie procesu uczenia zdecydowanie polepszyło rezultaty w niemalże każdej z analizowanych metryk. Agent zużywał mniej amunicji ponieważ jego celność się zwiększała, nagroda, killcount oraz episode length rosły. Na podstawie eksperymentów można wywnioskować, że model PPO radzi sobie lepiej niż A2C. Kluczem w osiągnięciu prawidłowego uczenia był odpowiedni dobór wartości w funkcji nagrody oraz trenowanie samymi obrazami przy pomocy sieci CNN. Pomysł na dodatkowe uczenie przy pomocy danych skalarnych okazał się nietrafiony. Jednocześnie nie ma pewności, czy przetwarzanie obrazu w postaci wykrywania krawędzi było istotnie potrzebne. Końcowe rezultaty są jednak zadowalające. W przypadku scenariusza basic agent radził sobie zdecydowanie lepiej niż prawdziwy gracz. Podobnie w defend center człowiek niewytrenowany nie byłby w stanie zabić tylu przeciwników. W przypadku scenariusza "Deadly Corridor" można polemizować, kto poradziłby sobie lepiej. Jednakże, w przeciwieństwie do człowieka, agent ma nieograniczone pole rozwoju oraz nie posiada barier biologicznych wpływających na szybkość reakcji i podejmowania decyzji.

7 Dalsze możliwości rozwoju

Wśród dostępnych scenariuszy ViZDoom wiele nie zostało jeszcze przetestowanych (deathmatch, labirynt). Tam można upatrywać polepszania modelu. Dodatkowo pojedynek agenta z człowiekiem mógłby dostarczyć informacji o rzeczywistym czasie reakcji i dokładności modelu, gdy przeciwnik jest mniej przewidywalny. Oczywiście nie mamy pewności czy obrane parametry funkcji nagrody są tymi najkorzystniejszymi. Dalszy rozwój mógłby pójść w kierunku ponownej optymalizacji, a w najlepszym przypadku zautomatyzowania tego procesu na obraz metodyk grid search.

Bibliografia

[1] John Schulman i in. *'Proximal Policy Optimization Algorithms'*. URL: `https://arxiv.org/abs/1707.06347`.

[2] Michał Kempka i in. *'ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning'*. URL: `https://arxiv.org/abs/1605.02097`.

[3] Thomas Simonini i in. *'The intuition behind PPO'*. Dostęp: 09.06.2024. URL: `https://huggingface.co/learn/deep-rl-course/unit8/intuition-behind-ppo`.

[4] Volodymyr Mnih i in. *Asynchronous Methods for Deep Reinforcement Learning*. URL: `https://arxiv.org/abs/1602.01783`.

[5] Richard S. Sutton i Andrew G. Barto. *'Reinforcement Learning: An Introduction'*. 2018.